(19) Bundesrepublik Deutschland
Deutsches Patent- und Markenamt

(10) **DE 103 47 975 A1** 2004.05.13

(12) # Offenlegungsschrift

(21) Aktenzeichen: **103 47 975.9**
(22) Anmeldetag: **15.10.2003**
(43) Offenlegungstag: **13.05.2004**

(51) Int Cl.[7]: **G11C 16/02**

(66) Innere Priorität:
102 49 676.5 24.10.2002

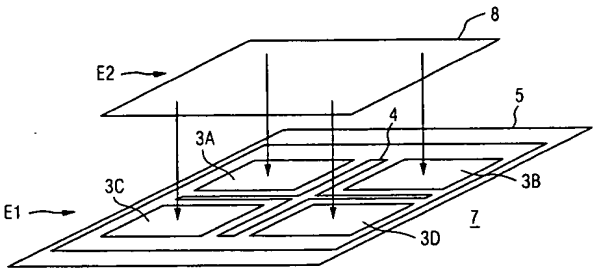
(71) Anmelder:
Siemens AG, 80333 München, DE

(72) Erfinder:
Siemers, Christian, Prof. Dr., 38446 Wolfsburg, DE

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Bezeichnung: **Einrichtung der programmierbaren Logik**

(57) Zusammenfassung: Die Einrichtung (7) der programmierbaren Logik umfasst mehrere Logikblöcke (3A bis 3D) mit konfigurierbaren Eigenschaften und Mittel zum Verknüpfen der Logikblöcke untereinander und mit einer Verarbeitungseinheit (4) und einer Ein-/Ausgabeeinheit (5). Eine Rekonfigurierbarkeit der Logikblöcke (3A bis 3D) während des Betriebs der Logikeinrichtung (7) ist dadurch gegeben, dass die Verknüpfungsmittel zusätzlich wenigstens einen konfigurierbaren Umschalt-Logikblock (8) aufweisen, mit dem die Konfiguration wenigstens einiger der rekonfigurierbaren Logikblöcke (3A bis 3D) untereinander und/oder mit der Verarbeitungseinheit (4) und/oder der Ein-/Ausgabeeinheit (5) erfolgt.



Beschreibung

[0001] Die Erfindung bezieht sich auf eine Einrichtung der programmierbaren Logik mit mehreren Logikblöcken mit konfigurierbaren Eigenschaften, die jeweils mindestens eine logische Verarbeitungseinheit mit Funktionsprogrammen und Schnittstellen zu den jeweils anderen Logikblöcken umfassen, mit mindestens einer den Logikblöcken zugeordneten Eingabe- und Ausgabeeinheit sowie mit Mitteln zum Verknüpfen der Logikblöcke untereinander, mit der mindestens einen Verarbeitungseinheit eines anderen Logikblocks und mit der mindestens einen Ein-/Ausgabeeinheit. Eine derartige Logikeinrichtung ist der US 4,870,302 A zu entnehmen.

[0002] Programmierbare logische Bausteine von herkömmlichen Logikeinrichtungen wie insbesondere Prozessoren führen Programme aus, die aus einem Speicher geladen werden. Die auszuführende Software in Form von Befehlswörtern ist dort als Maschinenbefehl abgelegt. Diese Befehlswörter werden geladen, analysiert, interpretiert und in einer Verarbeitungseinheit ausgeführt. Dabei löst die Verarbeitung eines einzigen Befehlswortes eine Vielzahl von Einzelaktionen in der Logikeinrichtung aus.

[0003] Die Basisstruktur und -organisation bekannter digitaler Logikeinrichtungen, insbesondere von Computern mit Mikroprozessoren, beruht auf der Konzeption des sogenannten „Von-Neumann-Rechners". Dessen Zentraleinheit CPU („Central Processing Unit"), d.h. dessen Computerkern, umfasst in ihrer Minimalkonfiguration als Hauptbestandteile einen Hauptspeicher, eine Steuereinheit und eine Verarbeitungseinheit (bzw. Rechenwerk):

- Der Hauptspeicher speichert Befehlswörter (Programmdaten) und Verarbeitungsdaten (Operantenwörter) und stellt diese auf Aufforderung zur Verfügung. Ferner nimmt der Hauptspeicher Zwischen- und Endresultate der Verarbeitung auf. Hauptspeicher können durch flüchtige oder nicht-flüchtige Speicher realisiert werden.
- Die Steuereinheit organisiert die Reihenfolge, in der Befehlswörter abgearbeitet werden. Sie fordert Befehlswörter aus dem Hauptspeicher an und veranlasst deren Ausführung in der Verarbeitungseinheit. Außerdem analysiert sie die Befehlswörter und veranlasst die Lieferung von Verarbeitungsdaten an die Verarbeitungseinheit.
- Die Verarbeitungseinheit führt die Operation an den Verarbeitungsdaten aus und liefert entsprechende Resultatwörter an den Hauptspeicher. Für jede Operation enthält die Verarbeitungseinheit ein Mikroprogramm, das die benötigten Übertragungswege freischaltet. Die Verarbeitungseinheit wird durch die Steuereinheit auf die jeweilige Operationsart, d.h. auf den abzuarbeitenden Befehl, eingestellt.

[0004] Der Zentraleinheit sind Peripheriegeräte zugeordnet, bei denen es sich um externe Speicher sowie um Ein- und Ausgabegeräte handeln kann. Die angegebenen Hauptkomponenten der Zentraleinheit können physikalisch getrennt sein; zumeist sind sie jedoch auf einem gemeinsamen Prozessorchip mit einem Cache oder beispielsweise einem embedded ROM realisiert.

[0005] Eine genauere Betrachtung der Programmierung einer solchen programmierbaren Logikeinrichtung PLD („Programmable Logic Device") mit den erwähnten Grundkomponenten des Von-Neumann Rechners zeigt, dass das Programm sowie die Initialisierungsdaten im preiswerten Speicher (mit minimal 1 Transistor pro Speicherzelle) untergebracht sind und erst dann in die CPU zur Ausführung gelangen, wenn sie an der Reihe sind. Die preiswerte Speicherung einerseits und die „Wiederverwendung" der aufwendigeren CPU-Schaltkreise wie ALU („Arithmetic Logic Unit") für alle möglichen Instruktionen andererseits sind die positiven Beiträge zu einer sogenannten „funktionalen Dichte" der Prozessor-basierten Rechner. Hierbei ergibt die zeitliche Sequenz der Bearbeitung natürlich einen negativen Beitrag.

[0006] Die funktionale Dichte kann dabei als mittlere Anzahl von aktiven Gatteräquivalenten pro Siliziumfläche und Zeit definiert werden.

[0007] Bei bekannten programmierbaren Logikeinrichtungen (PLDs) mit Rechenkapazität, Speicher und Ein- und Ausgabeeinheit I/O („Input/Output"), wie sie z.B. aus der eingangs genannten US-A-Schrift zu entnehmen ist, wird deren Struktur einmal programmiert, und der Programm-(wie Daten-)Inhalt wird an der Struktur selbst gespeichert. Diese Form der Speicherung ist aufwendig, denn um den Speicher selbst müssen zwecks schneller Umsetzung in geschaltete Datenpfade eine Menge von zusätzlichen Transistoren hinzukommen. Es existieren zwar nur wenige Angaben über die Ausnutzung des Siliziums; man sollte von Faktoren 20 bis 40 im Verhältnis Gesamtzahl/sichtbarer Transistorkapazität (im Sinne der Schaltfunktion) ausgehen. Bei der bekannten Ausführungsform einer Logikeinrichtung bezieht sich die Verknüpfung ihrer Logikblöcke erstens auf die Kopplung von Datenausgängen auf Dateneingänge (Routing-Verknüpfung) und zweitens auf die Verarbeitung der Eingangsdaten zu den Ausgangsdaten in den einzelnen Logikblöcken (Logik-Verknüpfung). Die Routing-Verknüpfung gilt sowohl für Daten, die aus Logikblöcken (Logic Elements) stammen bzw. in diese geführt werden, als auch für solche, die aus I/O-Pads stammen bzw. in diese geleitet werden. Es ist ausschließlich an eine Datenkopplung in diesem System gedacht.

[0008] Aus der US 6,333,641 B1 geht eine programmierbare Logikeinrichtung mit einem Array von Logikmodulen oder -blöcken hervor. Eine Verbindungseinheit mit vertikalen Routing(Leitweg-)Bahnen, horizontalen Routing-Bahnen und lokalen Routing-Bahnen verknüpft die Logikblöcke. Ein Omni-(universeller)Bus (Datenaustauschsammelschiene) ist über das Array gelegt, der mit dem Array derart verknüpft

ist, dass dieser dynamisch selbständige Sub-Arrays der Logikblöcke mit variabler Größe bildet, die ihrerseits mit dem Omni-Bus verbunden sind. Die Verknüpfung ist dabei von vornherein festgelegt. Auch hier handelt es sich wie im Fall der US 4,870,302 A um Datenverbindungen, d.h. um einen Austausch von Daten.

[0009] Da die Programmierbarkeit in Form eines Speichers mit Umsetzung in geschaltete Datenpfade recht teuer ist, wie im Fall des Rechners aber bislang das komplette Programm im Baustein lagert, erreichen die PLDs trotz Potenzials für hohe funktionale Dichte „nur" den Faktor 10 mehr als Prozessoren. Das bedeutet, dass der Geschwindigkeitsgewinn in PLDs gegenüber Prozessorlösungen teuer erkauft ist.

[0010] Aufgabe der vorliegenden Erfindung ist es deshalb, die programmierbare Logikeinrichtung mit den eingangs genannten Merkmalen dahingehend auszugestalten, dass bei hoher funktioneller Dichte eine hohe Geschwindigkeit des PLDs mit einfachen Mitteln zu erreichen ist.

[0011] Diese Aufgabe wird erfindungsgemäß mit den in Anspruch 1 angegebenen Maßnahmen gelöst. Diese Maßnahmen umfassen eine Rekonfigurierbarkeit der Logikblöcke während des gesamten Betriebs der Logikeinrichtung dadurch, dass die Verknüpfungsmittel zusätzlich wenigstens einen konfigurierbaren Umschalt-Logikblock aufweisen, mit dem eine Konfiguration wenigstens einiger der rekonfigurierbaren Logikblöcke selbst und/oder ihrer Verbindungen untereinander und/oder ihrer Verbindungen mit der Verarbeitungseinheit und/oder ihrer Verbindungen mit der Ein-/Ausgabeeinheit erfolgt, wobei der Umschalt-Logikblock in einer Ebene ausgebildet ist, die von einer Ebene mit den rekonfigurierbaren Logikblöcken verschieden ist. Unter einer verschiedenen Ebene des Umschalt-Logikblocks wird dabei jede Ebene verstanden, die nicht gleichzeitig die Ebene der rekonfigurierbaren Logikblöcke ist. D.h., die Ebene des Umschalt-Logikblocks kann über, neben oder unter der Ebene der rekonfigurierbaren Logikblöcke liegen.

[0012] Die mit einer solchen Architektur verbundenen Vorteile sind insbesondere darin zu sehen, dass bei begrenzter Anzahl von nunmehr rekonfigurierbaren Logikblöcken neben einer Konfiguration der Blöcke selbst auch die Verbindungen zwischen diesen Blöcken und/oder zu externen Bausteinen wie Speichereinheiten oder Mikroprozessoren wie z.B. zu der Verarbeitungseinheit und/oder zu der Ein-/Ausgabeeinheit nicht ein für alle Male fest vorgegeben sind, sondern dass diese Verbindungen mit Hilfe des zusätzlich vorgesehenen Umschalt-Logikblocks während des gesamten Betriebs bedarfsmäßig erstellt, d.h. konfiguriert werden können. Eine derartige Konfiguration mittels des Umschalt-Logikblocks kann als eine Verknüpfungsoperation der erfindungsgemäß ausgestalteten Verknüpfungsmittel angesehen werden.

[0013] Die Konfiguration kann dabei jederzeit, d.h. während der gesamten, ununterbrochenen Betriebsdauer – also nicht nur während einer Start- oder Boot-Phase – vorgenommen werden. Die erfindungsgemäßen Maßnahmen bedeuten also eine Verbindung von dem Umschalt-Logikblock zu Verknüpfungsbereichen und damit zu einer entsprechenden Konfiguration derselben. Damit wird es ermöglicht, dass von einzelnen Blöcken unterschiedliche Funktionen zu unterschiedlichen Zeitpunkten auszuführen sind; d.h., deren Ausnutzung wird entsprechend erhöht. Damit verbunden ist eine entsprechende Performance-Verbesserung der gesamten Logikeinrichtung, verglichen mit einem Mikroprozessor, bzw. kein Performance-Verlust im Vergleich zu herkömmlichen PLDs.

[0014] PLDs erhalten – wie von-Neumann-Prozessoren – zwei Arten von Informationen, Code und Daten. Der Code, der grundsätzlich die Aktionswoche bestimmt und bei PLDs Konfiguration bzw. Konfigurationscode genannt wird, wird üblicherweise vor dem eigentlichen Betrieb geladen und ist dann während des Betriebs unveränderlich. Die Konfiguration bestimmt u.a. die im Baustein aktiven Verbindungen.

[0015] Die Daten können sich während des Betriebs verändern und dadurch auch den aktuellen Verlauf der Operationen beeinflussen. Während also der Code alle möglichen Wegen beinhaltet, wird die tatsächliche Nutzung – dies entspricht dem aktuell durchlaufenden Pfad – (auch) von den Daten bestimmt.

[0016] Der Umschalt-Logikblock nach der Erfindung erhält ebenfalls Code und Daten. Das Wesentliche der Erfindung besteht nun darin, die übrigen, Nicht-Umschalt-PLDs, zu steuern, und zwar durch Wechseln bzw. Modifizieren des Codes.

[0017] Bei dem genannten Stand der Technik sowie bei den in der Praxis üblichen PLDs werden konfigurierbare Blöcke dadurch miteinander verbunden, dass die (Laufzeit-variablen) Daten austauschbar sind, also etwa in der Form, dass Datenausgänge des einen Blocks mit Dateneingängen des anderen verbunden sind. Bei dem erfindungsgemäßen Umschalt-Logikblock ist es jedoch so, dass die Ausgänge dieses Logikblocks zumindest partiell an den in anderen Ausführungsformen im Betrieb unzugänglichen Codebereich der konfigurierbaren Logikblöcke angeschlossen sind. Demgegenüber bezieht sich der Stand der Technik auf Ausführungsformen, bei denen die konfigurierbaren Logikblöcke im Datenpfad koppeln, nicht jedoch auf den Code Einfluss haben.

[0018] Vorteilhafte Ausgestaltungen der erfindungsgemäßen Logikeinrichtung gehen aus den abhängigen Ansprüchen hervor.

[0019] So kann insbesondere die Konfiguration wenigstens einiger der rekonfigurierbaren Logikblöcke einem vorgegebenen Kontext entsprechend erfolgen.

[0020] Ferner kann vorteilhaft der Umschalt-Logikblock wenigstens einen Zustandsspeicher aufwei-

sen, der Informationen bezüglich der Funktionen der einzelnen rekonfigurierbaren Logikblöcke enthält, so dass die Konfiguration der ausgewählten rekonfigurierbaren Logikblöcke gemäß den Funktionsinformationen des aktuellen Zustands erfolgt.

[0021] Die den erfindungsgemäßen Maßnahmen zu Grunde liegenden Überlegungen werden nachfolgend unter Bezugnahme auf die Zeichnung noch weiter erläutert. Dabei zeigen

[0022] deren **Fig.** 1 die Grundstrukturen von vier einfachen, bekannten Automatentypen,

[0023] deren **Fig.** 2 die Grundstruktur einer sogenannten „Sequential Finite State Machine",

[0024] deren **Fig.** 3 den schematischen Aufbau einer partiell rekonfigurierbaren PLD

[0025] und deren **Fig.** 4 den schematischen Aufbau einer erfindungsgemäßen Logikeinrichtung.

[0026] Im Folgenden werden aus den Maschinen vom sogenannten „Finite State"-Typ die sogenannten „Sequential Finite State Machines" (SFSM) abgeleitet. Das hierfür aufgezeigte Modell ist dazu geeignet, eine Sequenz von Konfigurationen im PLD zu definieren, und genau diese Sequenz kann ohne Performance-Verlust eine wesentlich höhere funktionale Dichte (gleichbedeutend mit drastisch gesenkten Kosten für Herstellung und Betrieb) eines PLDs erzeugen.

Abschnitt I („Sequential Finite State Machines")

[0027] Eines der „klassischen" Denkmodelle für eine Hardwareentwicklung besteht in den einfachen endlichen Automaten, im Folgenden „Finite State Machines" (FSM) genannt.

[0028] Dieses eng mit theoretischen Konzepten (insbesondere des sogenannten „Deterministischen endlichen Automaten") verwandte Modell ist exakt wie folgt definiert:

Definition 1:

[0029] Eine FSM besteht aus einem 6-Tupel {A, X, Y, f, g, a_0}.

$A = \{a_0, a_1, \ldots a_M\}$ ist hierbei die endliche Menge der Zustände, wobei a_0 den Startzustand bedeutet.

$X = \{X_1, \ldots X_K\}$ ist die endliche Menge der Eingangsvektoren mit $X_I = (x_1, \ldots x_L)_i$, wobei $x_1 \in \{0, 1, -\}$.

$Y = \{Y_1, \ldots Y_N\}$ ist die endliche Menge der Ausgangsvektoren mit $Y_J = (y_1, \ldots y_H)_j$, wobei $y_h \in \{0, 1, -\}$.

$f: A \times X \rightarrow A$ heißt Transitionsfunktion (Next State Decoder),

$g: A \times X \rightarrow Y$ heißt Ausgangsfunktion (Output Decoder), t = Zeiteinheit.

[0030] Die in dieser Definition genannten Funktionen werden durch Schaltnetze realisiert, die den algorithmischen Zusammenhang zwischen den (im Wesentlichen binären) Eingangs- und Zustandsvektoren darstellen. Hierzu ist allerdings notwendig, dass auch die Zustände binär codiert werden, was in Definition 1 noch nicht der Fall war.

[0031] Für die Zustandsautomaten werden weiterhin drei wesentliche Untertypen (siehe **Fig.** 1) unterschieden, deren Einfluss auf die Komplexität der Funktionen in der Praxis gegeben ist. Ein Entwickler wird folgende Schritte durchführen:

1. Festlegen der Signale: Das Schaltwerk wird als „Black Box" mit den erforderlichen Eingangs- und Ausgangssignalen skizziert.
2. Entwerfen des Zustandsdiagramms: Dieser Schritt ist der eigentliche Kern der Synthese, da hier das zu lösende Problem formal beschrieben wird. Als Zustandsdiagramm sind Zustandsgraphen, Programmablaufpläne oder auch Schaltwerktabellen möglich.
3. Aufstellen der Schaltwerktabelle als formaler Ausgangspunkt für alle weiteren Operationen.
4. Zustandsminimierung: Die Minimierung der Anzahl der Zustände soll eine Vereinfachung des Designs erreichen.
5. Zustandscodierung: Bei synchronen Schaltwerken mit synchronisierten Eingängen (nicht für Mealy-Automat gemäß **Fig.** 1d) lassen sich beliebige Codierungen für die Zustände aus Z angeben. Beim Medwedjew-Automat gemäß **Fig.** 1b müssen die Codierungen allerdings mit den gewünschten Ausgangssignalen übereinstimmen; beim Moore-Automat gemäß **Fig.** 1c hingegen können die Codierungen so gewählt werden, dass sich vereinfachte Schaltnetze für Next State Decoder (f) und Output Decoder (g) ergeben.
6. Berechnung von Folgezustands- und Ausgangsschaltnetzen:
   Zur konkreten Berechnung des Folgezustandsschaltnetzes muss ein Register- bzw. Flipfloptyp gewählt werden, da deren Eingänge (T, D, RS oder JK) durch dieses Schaltnetz belegt werden, aber unterschiedliche Funktionalitäten zeigen.
7. Realisierung des Schaltwerks und Test

[0032] Aus der Praxis weiß man nun, dass die Implementierung als einfache Maschine – man könnte sie auch als „flache Maschine" bezeichnen – nicht unbedingt die beste ist. Ein kooperierender Automat, bestehend aus mehreren einfachen Automaten, die miteinander gekoppelt sind, kann dies häufig wesentlich besser im Sinne von flächeneffizienter. Zudem sollte bedacht werden, dass der vorstehend skizzierte Designfluss nicht unbedingt in der angegebenen Weise durchgeführt werden muss. Gerade mit zunehmender Beschreibung in Hochsprachen wie VHDL ist eine Hinwendung zu mehr algorithmischem Stil zu erkennen.

[0033] Unabhängig davon soll jedoch die FSM als das grundlegende Modell angenommen werden. Man kann sich nun vorstellen, nicht nur eine FSM zu haben, sondern mehrere, von denen exakt eine zu einem Zeitpunkt aktiv ist. Ein ausgezeichneter Teil (der sogenannte „Sequencer") schaltet dann in Abhängigkeit von Eingangssignalen oder erreichten Resultaten zwischen den einzelnen FSMs um.

[0034] Ausgehend von dieser FSM wird nachfolgend die Sequential Finite State Machine (SFSM) korrekt definiert und dargestellt

Definition 2:

[0035] Eine Sequential Finite State Machine (SFSM) besteht aus einem 5-Tupel (B, B_0, C, V, h) . Hierbei stellt B = {B_0, ... B_K} eine endliche Menge von Finite State Machines (FSM) dar, B_0 ist die Start-FSM. C = {C_0,... C_K} beschreibt eine endliche Menge von Zuständen zur Kenntlichmachung der aktuellen FSM. V = {V_1,.. V_N} ist die endliche Menge der (zusätzlichen) Eingangsvektoren mit V_I = (v_1,... v_L)$_i$, wobei $v_1 \in \{0, 1, -\}$. h: B × V → B heißt FSM-Transitionsfunktion (Next FSM Decoder).

[0036] Der wesentliche Vorteil dieser SFSM (vgl. hierzu auch **Fig.** 2) liegt erst einmal in der Modellierung. Ein Designer hat die Chance, sein Design in kleinere Portionen zu teilen. In der technischen Ausführung wird man dann versuchen, die (weiterhin endliche) Menge der Zustände aus allen FSMs B_k auf eine einheitliche Zustandscodierung abzubilden. Zusätzlich hierzu muss eine Codierung für die C_k der aktuellen FSM B_k mitgeführt werden, um für die Funktion h die Berechnung der nächsten FSM zu ermöglichen.

[0037] Andererseits zeigt ein Blick auf **Fig.** 2, dass eigentlich nur wenig gewonnen ist. Falls der Takt für alle Register identisch ist, dann wurden in **Fig.** 1a die Register und das Schaltnetz f(u*, x*) (hext State Decoder), * bedeutet Gesamtmenge) in Teilmengen von Registern mit f(u, x, c) und h(u, v, c) eingeteilt, die in der Realisierung wieder zusammengeführt werden. Minimaler wird die Lösung dadurch kaum, denn in einem PLD heutiger Bauart müssen alle Teile, also alle Teil-FSM, auf dem Baustein integriert werden. Hier setzen nun die erfindungsgemäßen Überlegungen an.

[0038] Es existieren zwar wenige FPGAs („Field-Programmable Gate Arrays"; vgl. z.B. „Spektrum der Wissenschaft", August 1997, Seiten 44 bis 49), die dynamisch rekonfigurierbar sind, jedoch nur partiell. Diese rekonfigurierbare Eigenschaft könnte man ausnutzen, indem in einem permanenten Teil h(u, v, c) und die Register zur FSM-Codierung sowie im nachladbaren Teil die aktuelle FSM geladen und ausgeführt werden. Das Problem hierbei ist der Performanceverlust beim Nachladen. Dies ist aus dem nachfolgenden Beispiel ersichtlich:

Beispiel

[0039] Die gern genommene Straßenverkehrsampel als Beispiel für eine Finite State Machine kann auch sehr gut als Beispiel für eine Sequential FSM genommen werden, wenn man an einen Nachtmodus denkt. Das Wort Modus zeigt schon den Weg: Im Umschalten zwischen einzelnen FSMs sollte so etwas wie ein Moduswechsel liegen, und Tag- und Nachtmodus schließen sich nun einmal gegenseitig aus.

[0040] FSM1 integriert nun die Tagampel, FSM2 die Nachtampel (z.B. gelbes Blinken für die Nebenstraßen), und im übergeordneten Sequencer wird an Hand eines Zeitsignals entschieden, welche FSM zur Ausführung kommt und welche ruht. Der Sequencer wird als FSMO implementiert.

[0041] Die Beschreibung für hierfür erforderliche drei FSM (0 .. 2) wird getrennt durchgeführt, dann aber auf einem PLD integriert, so dass im Bereich 0 die FSMO, 1 FSM1 und 2 FSM2 liegen würde. Ein Flächengewinn wäre damit nicht zu verzeichnen.

[0042] Im nachladbaren Fall müsste FSMO permanent zur Verfügung stehen, außerdem ein Bereich, der das Maximum von {FSM1, FSM2} aufnehmen könnte. In diesen Bereich wäre dann ständig eine von zweien geladen, und das Umschalten würde eine Rekonfiguration nach sich ziehen.

[0043] Zur Verdeutlichung des Vorteils bei dynamisch rekonfigurierbaren PLDs, wie sie Logikeinrichtungen nach der Erfindung darstellen, wird die Übergangsfunktion f(u, x, c) (vgl. Figur 2) nunmehr als f_c(u, x) (Äquivalentes gilt für Funktion g) bezeichnet. Mit dieser Darstellung soll die Unterschiedlichkeit in der Abhängigkeit demonstriert werden: u und x bewirken eine ständige, mit dem Takt verbundene Fortentwicklung des Zustands der Maschine, während c die Abhängigkeit vom wesentlich selteneren Moduswechsel beschreibt.

[0044] Die Rekonfiguration selbst würde vergleichsweise viel Zeit in Anspruch nehmen; es wurde nun erkannt, dass eine wesentlich günstigere Lösung in einer erweiterten Architektur von PLDs besteht, in denen zugleich mehrere Programme dynamisch umschaltbar gespeichert sind. Eine entsprechende Logikeinrichtung könnte als Multi-Plane-PLD bezeichnet werden; um jedoch dem Aspekt des jeweiligen „Kontext" Rechnung zu tragen und darzustellen, dass auch nur Teilbereiche von Umschaltvorgängen betroffen sind, wird hier der Name „Dynamical Multi-Context PLDs (dMC-PLDs)" gewählt.

Abschnitt II (Erfindungsgemäße Architektur von „Dynamical Multi-Context PLDs")

[0045] Zunächst sei ein Problem angesprochen, dass sowohl für klassische dynamisch rekonfigurierbare als auch für dMC-PLDs gilt. Wird nämlich lediglich ein Teilbereich umgeschaltet, müssen dessen Schnittstellen nach außen hin, also insbesondere in Richtung der verbleibenden Teile, konstant bleiben. Zugleich sind einige Funktionsprogramme (sogenannte „Routingressourcen") von dem Umschalten mit betroffen, da innerhalb des umzuschaltenden Blocks nicht nur Logik, sondern auch Verbindungen liegen müssen.

[0046] Konsequenterweise wird im Folgenden für die erfindungsgemäße Logikeinrichtung ein PLD-Block als eine rekonfigurierbare Einheit, beste-

hend aus logischen Verarbeitungseinheiten mit Routingressourcen und Schnittstellen (sogenanntes „Routing") zu anderen Blöcken definiert; diese Definition stimmt mit den in der Praxis üblichen, partiell rekonfigurierbaren Bausteinen überein. Die Definition dieses Blocks weicht dabei von der üblichen Definition der logischen Blöcke ab.

[0047] Wenn man sich jetzt einmal einen fiktiven PLD mit solchen Blöcken anschaut, erhält man eine Architektur, wie sie aus **Fig.** 3 hervorgeht. In der Figur sind bezeichnet mit 2 eine partiell rekonfigurierbare PLD-Einrichtung bzw. -Struktur, mit 3A bis 3D vier PLD-Blöcke (= Logikblöcke), mit 4 ein (globales) Routing, das eine Verarbeitungseinheit mit Funktionsprogrammen und Schnittstellen zu den Blöcken bildet, sowie mit 5 ein I/O-Bereich, der Mittel zum Verknüpfen der Logikblöcke mit der Verarbeitungseinheit und einer Ein-/und Ausgabeeinheit darstellt. Man kann sich diese Architektur so vorstellen, dass – einmal abgesehen von den I/O-Ressourcen **5** am Außenrand – jeder PLD-Block **3A** bis **3D** einem vollwertigen CPLD („Complex PLD") oder FPGA entspricht. Derartige Architekturen existieren bereits, z.B. „Cypress Delta39k", „Lattice-Vantis Godfather"-Architektur; sie sind jedoch nicht im Betriebszustand der PLD-Einrichtung rekonfigurierbar.

[0048] Bei der in **Fig.** 4 schematisch angedeuteten dMC-PLD-Architektur einer Logikeinrichtung **7** nach der Erfindung wird die PLD-Struktur **2** nach **Fig.** 3 durch einen zusätzlichen PLD überlagert. Dieser als Umschalt-Logikblock **8** bezeichnete Extra-PLD ist bei bekannten Logikeinrichtungen nicht vorhanden und soll die Umschaltfunktion h(u, v, c) (vgl. **Fig.** 2) übernehmen. Er stellt folglich einen Teil von Verknüpfungsmitteln dar, mit denen die Konfiguration wenigstens einiger der Logikblöcke **3A** bis **3D** untereinander und/oder mit der Verarbeitungseinheit **4** und/oder der Ein-/Ausgabeeinheit **5** erfolgt. Dies bedeutet, dass die Ausgänge in **Fig.** 4 einem gespeicherten Kontext c entsprechen und die Auswahl der aktuellen Funktionen $f_c(u, x)$ und $g_c(u, x)$ steuern.

[0049] Für diesen Extra-PLD ergibt sich damit eine Struktur, die aus vielen Eingängen (u, x) wenige, ausschließlich gespeicherte Ausgänge berechnet.

[0050] Die sich aus diesem Aufbau ergebenden Vorteile sind darin zu sehen, dass die funktionale Dichte und damit die Ausnutzbarkeit gesteigert werden bzw. die Kosten pro Anwendung zu senken sind. Folgende grobe Abschätzung kann dafür gegeben werden:

Es sei angenommen, dass die Ausnutzung der verwendeten Transistoren 1:20 ist; d.h., von 20 eingesetzten Transistoren ist tatsächlich nur einer in der (programmierbaren) logischen Funktion sichtbar. Für den Extra-PLD **8** lassen sich 20 % Overhead veranschlagen, da es sich um einen Logikblock handelt, der nicht zur eigentlichen Funktionalität beiträgt, sondern nur die anderen Blöcke **3A** bis **3D** umschaltet. Für die Speicherung und Decodierung der zusätzlichen Programme (es werden lediglich 4 gespeicherte Funktionen pro PLD-Block angenommen) werden weitere 6 Transistoren berechnet (4 Transistoren zur Speicherung und 2 zur Auswahl 1 aus 4). Daraus ergibt sich eine neue Ausnutzung von 4:30, mithin eine Steigerung der Dichte gegenüber bekannten Ausführungsformen um 166 %.

[0051] Gegebenenfalls könnte sich eine Schwierigkeit dadurch ergeben, dass sich auf Grund mehrerer Konfigurationen, aus denen eine aktuelle auszuwählen ist, der kritische Pfad innerhalb eines PLD-Blocks um einen Transistor verlängern kann, was mit der Erniedrigung der maximalen Taktfrequenz einhergehen würde. Hierzu muss zwischen sogenannter SRAM-basierten Speicherung einerseits und sogenannter FLASH-EEPROM bzw. Anti-Fuse andererseits unterschieden werden.

[0052] Für SRAM-basierte Verfahren wird mit der US 6,011,740 A ein Verfahren vorgeschlagen, die Speicherung mehrerer Konfigurationen in einem Schatten-RAM z.B. als Ringspeicher zu realisieren und die aktuell benötigte Information in eine Arbeitszelle zu laden. Dieses Verfahren garantiert eine maximale Arbeitsgeschwindigkeit.

[0053] Für Flash- und Antifuse-Speicherung könnte als eigentliche Zelle ebenfalls eine nachladbare SRAM-Zelle in Betracht kommen, so dass die obengenannte Lösung zum Tragen kommt; dies bedeutet jedoch einen zusätzlichen Aufwand und möglicherweise eine Verringerung der Geschwindigkeit, bedingt durch den Technologiewechsel bei der Speicherung. Bei diesen Technologien wird tatsächlich ein weiterer Path-Transistor, der den aktuellen Kontext decodiert, im Pfad eingeführt, gegebenenfalls in Form von Dual-Gate-FETs.

[0054] Die genaue Einteilung, welche Blockgröße und Extra-PLD-Größe für die Vielzahl an Anwendungen von Vorteil sind, kann durch Simulationen an Beispielen ermittelt werden.

[0055] Zu einer Realisierung einer erfindungsgemäßen Logikeinrichtung nach **Fig.** 4 kann man sich vorstellen, dass die beiden schematisch dargestellten Schichten, nämlich die mit dem Extra-PLD **8** (= Umschalt-Logikblock) und die mit den Logikblöcken **3A** bis **3D**, jeweils in einer Ebene E2 bzw. E1 zumindest weitgehend äquivalent sind und sich nicht nur gegenseitig umschalten, sondern vielmehr auch die Programmierung des jeweils anderen Teils erzeugen. Umschaltung bedeutet in diesem Fall, dass zwischen vorbestimmten Programmen ausgewählt wird. Gegenseitige Programmierung hingegen kann zur Anpassung (Evolution) an die jeweilige Umgebung genutzt werden. Ein einfaches Beispiel kann eine digitale PLL sein, die eine Frequenz mittels Laufzeiteffekten an einen Mastertakt anpasst.

Patentansprüche

1. Einrichtung der programmierbaren Logik
– mit mehreren Logikblöcken mit konfigurierbaren Eigenschaften, die jeweils mindestens eine Verarbei-

tungseinheit mit Funktionsprogrammen und Schnittstellen zu den jeweils anderen Logikblöcken umfassen,
– mit mindestens einer den Logikblöcken zugeordneten Ein-/Ausgabeeinheit,
und
– mit Mitteln zum Verknüpfen der Logikblöcke
a) untereinander,
b) mit mindestens einer der Verarbeitungseinheiten eines anderen Logikblocks
und
c) mit der mindestens einen Ein-/Ausgabeeinheit, gekennzeichnet durch eine Rekonfigurierbarkeit der Logikblöcke (**3A** bis **3D**) während des gesamten Betriebs der Logikeinrichtung (**7**) dadurch, dass die Verknüpfungsmittel zusätzlich wenigstens einen konfigurierbaren Umschalt-Logikblock (**8**) aufweisen, mit dem eine Konfiguration wenigstens einiger der rekonfigurierbaren Logikblöcke (**3A** bis **3D**) selbst und/oder ihrer Verbindungen untereinander und/oder ihrer Verbindungen mit der mindestens einen Verarbeitungseinheit (**4**) und/oder ihrer Verbindungen der mindestens einen Ein-/Ausgabeeinheit (**5**) erfolgt, wobei der Umschalt-Logikblock (**8**) in einer Ebene (E2) ausgebildet ist, die von einer Ebene (E1) mit den rekonfigurierbaren Logikblöcken (**3A** bis **3D**) verschieden ist.

2. Einrichtung nach Anspruch 1, dadurch gekennzeichnet, dass die Ebenen (E1, E2) zumindest weitgehend äquivalent aufgebaut sind.

3. Einrichtung nach Anspruch 1, dadurch gekennzeichnet, dass die Konfiguration wenigstens einiger der rekonfigurierbaren Logikblöcke (**3A** bis **3D**) einem vorgegebenen Kontext (c) entsprechend erfolgt.

4. Einrichtung nach einem der vorangehenden Ansprüche, dadurch gekennzeichnet, dass der Umschalt-Logikblock (**8**) wenigstens einen Zustandsspeicher aufweist, der Informationen bezüglich der Funktionen der einzelnen rekonfigurierbaren Logikblöcke (**3A** bis **3D**) enthält, und dass die Konfiguration der ausgewählten rekonfigurierbaren Logikblöcke gemäß den Funktionsinformationen des ausgewählten Zustands erfolgt.

Es folgen 3 Blatt Zeichnungen

Anhängende Zeichnungen

FIG 1a

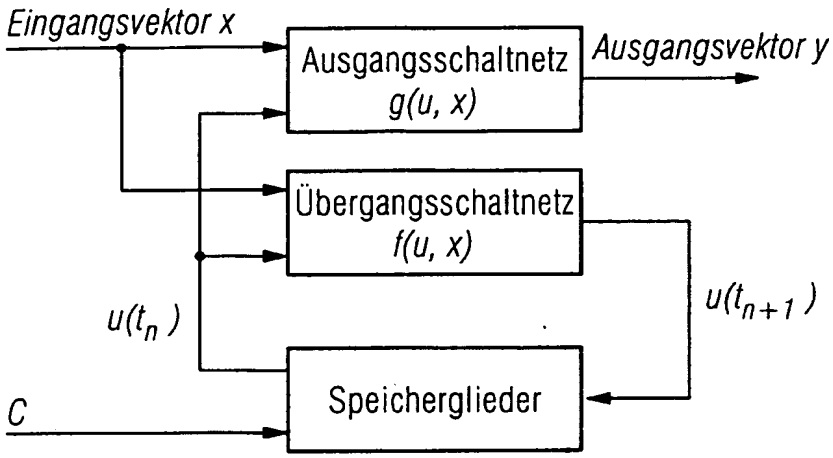


FIG 1b

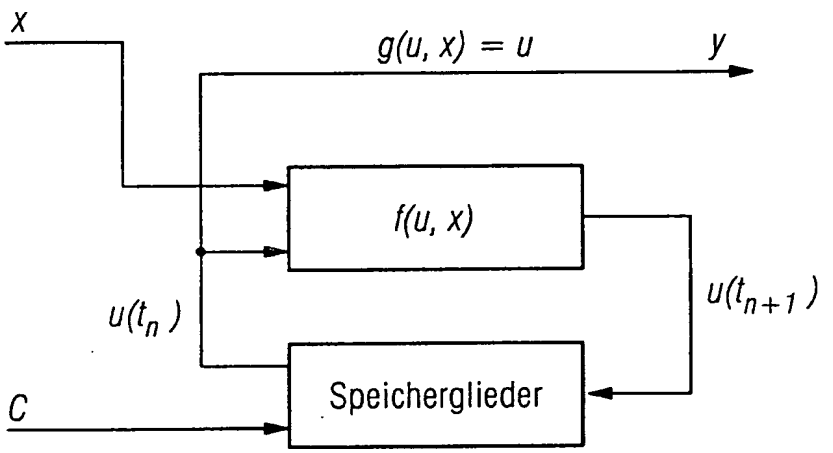


FIG 1c

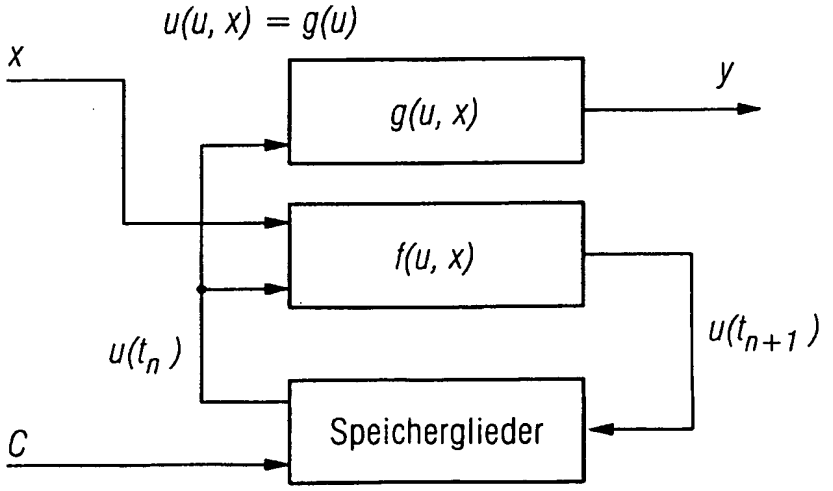


FIG 1d

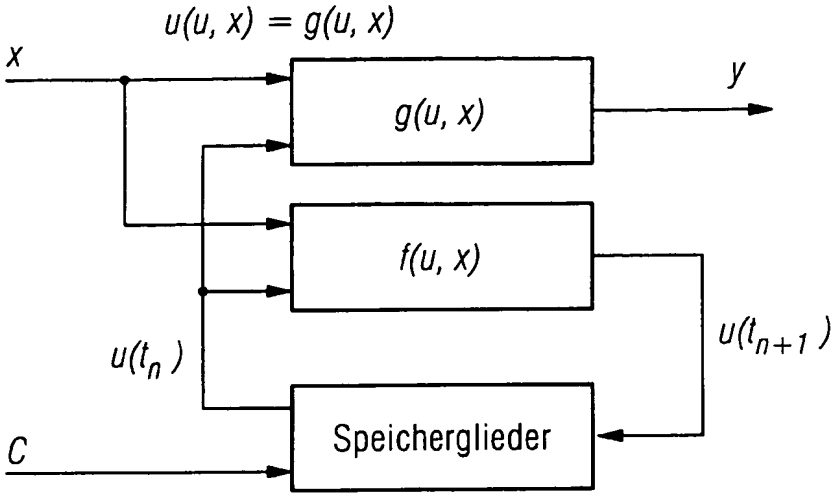

u(u, x) = g(u, x)
x
y
g(u, x)
f(u, x)
u(t_n+1)
u(t_n)
C
Speicherglieder

FIG 2

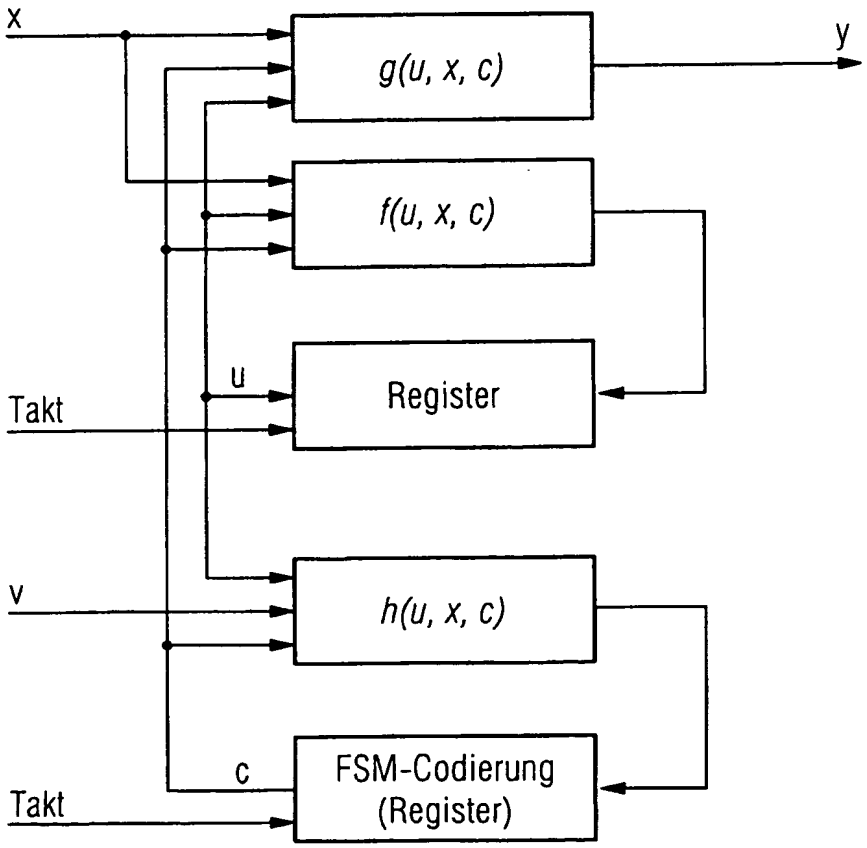

x
y
g(u, x, c)
f(u, x, c)
u
Takt
Register
v
h(u, x, c)
c
Takt
FSM-Codierung
(Register)

FIG 3

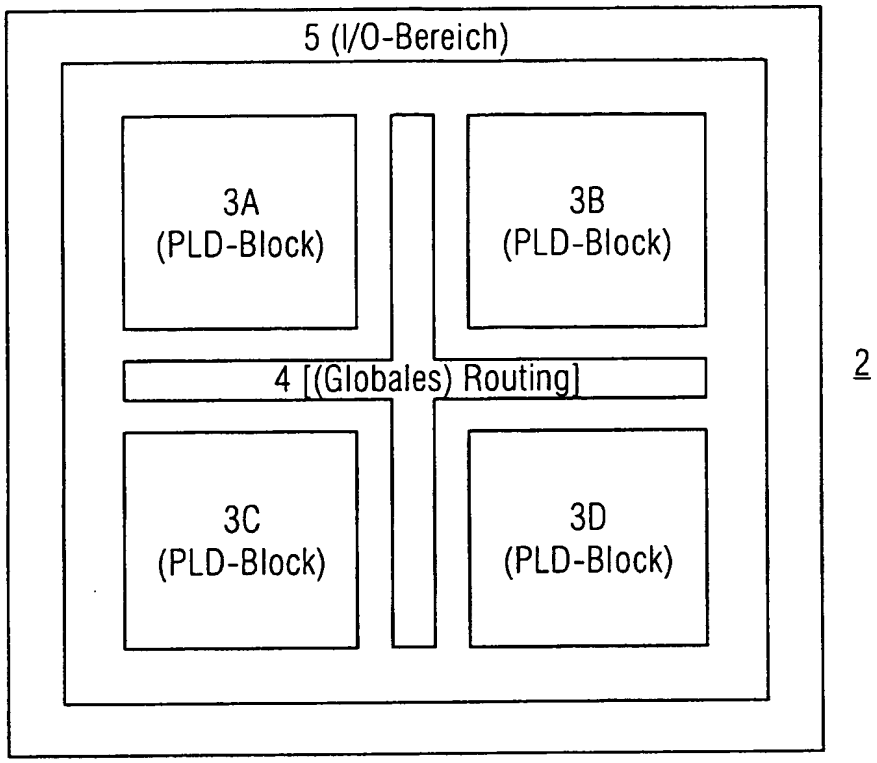
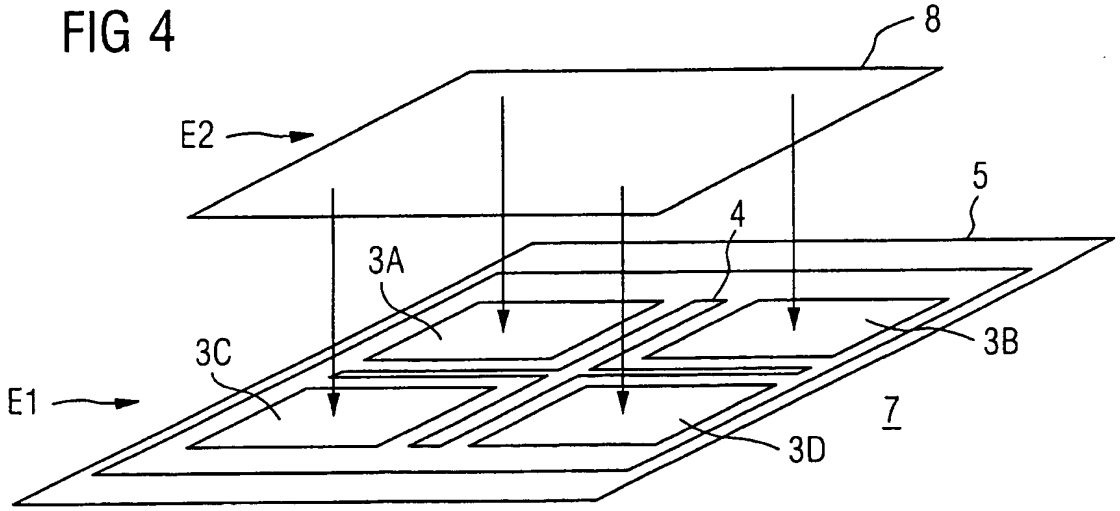

5 (I/O-Bereich)
3A
(PLD-Block)
3B
(PLD-Block)
4 [(Globales) Routing]
3C
(PLD-Block)
3D
(PLD-Block)
2

FIG 4


8
E2
5
4
3A
3C
3B
E1
7
3D

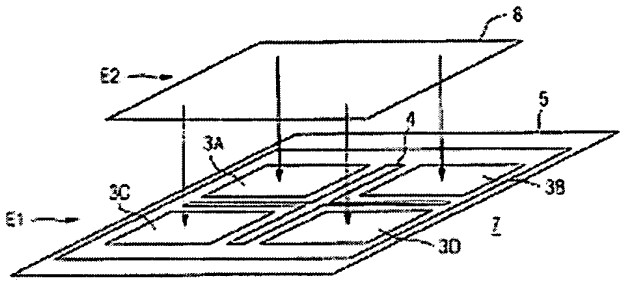
Programmable logic device using logic blocks with variable configuration determined by configurable switching logic block

Patent number:	DE10347975
Publication date:	2004-05-13
Inventor:	SIEMERS CHRISTIAN (DE)
Applicant:	SIEMENS AG (DE)
Classification:	
- international:	***H03K19/177;*** **H03K19/177;** (IPC1-7): G11C16/02
- european:	H03K19/177
Application number:	DE20031047975 20031015
Priority number(s):	DE20031047975 20031015; DE20021049676 20021024

Report a data error here

Abstract of **DE10347975**

The logic device (7) has a number of logic blocks (3A-3D) with configurable characteristics, each having at least one processing unit with function programs and interfaces for connection with each other logic block, at least one of the logic blocks having an input/output unit. The configuration of the logic device is determined by a configurable switching logic block (8) provided in a different plane (E2) to the plane (E1) containing the logic blocks.



Data supplied from the ***esp@cenet*** database - Worldwide